09/240509

METHOD AND SYSTEM FOR CLIENT-SERVER INTERACTION IN INTERACTIVE COMMUNICATIONS

Technical Field

The present invention relates to techniques for performing client-server interaction in communication systems and, more particularly, in communication systems based on the MPEG-4 standard.

Background of the Invention

Interactivity is a prominent concern in the development of the MPEG-4 international standard (ISO/IEC 14496 Parts 1-6, Committee Draft, October 31, 1997, Fribourg, Switzerland). A back channel is specified for interactive message support. However, the syntax and semantics of the messages to be carried through that channel remain unspecified, and so does the mechanism that triggers the transmission of such messages. Existing standards such as DSM-CC (ISO/IEC International Standard 13818-6) and RTSP (RFC 2326) support traditional VCR-type interactivity to reposition a media stream during playback, but this is inadequate for MPEG-4 applications which require more complex interactive control.

An interactive message can be generated by a certain user action or system event. It will then be sent to the server which in turn may modify the stream(s) it is delivering by adding or removing objects, or switching to an entirely new scene. User actions may include clicking on an object, input of a text stream, etc. System events include timers, conditional tests, etc.

Interactivity is application-specific, and one cannot define interactive behavior completely in terms of user events. To support application-specific interactivity, a CGI-like approach should be adopted. Specific user events cause application-specific command data to be sent back to the server. The server can then respond, typically by sending a scene description update command. This allows complete freedom for supporting full interactivity as may be required by applications.

MPEG-4 essentially uses two modes of interactivity: local and remote. Local interactivity can be fully implemented using the native event architecture of MPEG-4 BIFS (Binary Format for Scenes), which is based on the VRML 2.0 ROUTEs design (see www.vrml.org and "The VRML Handbook", J. Hartman and J. Wernecke, Addison-Wesley, 1996) and documented in Part 1 of the MPEG-4 specification (Systems). If the MPEG-4 receiver is hosted in another application, events that need to be communicated to the MPEG-4 receiver by the application can be translated to BIFS update commands, as defined in Part 1 of MPEG-4.

Remote interactivity currently consists of URLs. As defined in the MPEG-4 Systems Committee Draft, these can only be used to obtain access to content. As a result, they cannot be used to trigger commands.

The fact that MPEG-4 Systems already contains local interactive support via the use of event source/sink routes that are part of the scene description (BIFS) makes it desirable to have a server interaction process that fully integrates with the local interactivity model.

Summary of the Invention

An objective of the present invention is to provide a technique for communicating messages between two entities such as "client" and "server" utilizing the MPEG-4 international standard.

A second objective of the present invention is to provide a technique for allowing the user or the system to generate such messages in the context of an MPEG-4 player or client.

A third objective is to provide a technique for generating such messages consistent with the local interactivity model defined in MPEG-4, which is based on the VRML 2.0 specification.

A further objective is to provide a technique for encoding such messages within an MPEG-4 bitstream, as well as to link the encoded messages to the scene description.

Still a further objective is to provide a technique for encoding such messages in a way that allows a server to easily modify them before sending them for use by the client. This is important for interactive applications. An example is "cookie" management where a server must be able to quickly update the content of the command with a codeword that stores state information about the user's activities on the particular site.

In order to meet these and other objectives which will become apparent with reference to further disclosure set forth below, the present invention broadly provides a technique for incorporating server commands into MPEG-4 clients. The technique involves the use of command descriptors, i.e. a special form of descriptors that are transmitted together with the scene description information and contain the command to be sent back to a server upon triggering of an associated event. The desired event sources in the scene description are associated with these command descriptors.

In one embodiment, the association is performed using server routes. These operate similarly to traditional MPEG-4 BIFS routes, but instead of linking a source field with a sink field they link a source field with a sink command descriptor. Server routes require an extension of the MPEG-4 BIFS ROUTE syntax.

In another embodiment, the association is performed using command nodes. Such nodes contain sink fields, and are associated with command descriptors. This technique involves the addition of one more node type to the set of MPEG-4 BIFS nodes.

In both cases, the normal interaction model defined by MPEG-4 can be used

for both local interactivity, i.e. events generated and processed on the local clients, as well as server interactivity, i.e. as events generated on the client generate commands that are sent back to the server. Upon triggering of an event associated with a command descriptor, either via a server route or a regular route to a command node, the client obtains the command information stored in the command descriptor, packages it into a command message, preferably using the syntax provided in the preferred embodiment, and transmits it back to the server using the appropriate back channel.

The data to be carried by the generated command back to the server are contained in the command descriptor. Since command descriptors are part of the overall descriptor framework of MPEG-4, they can be dynamically updated, using time stamped object descriptor updates. This provides considerable flexibility in customizing commands, for example to perform "cookie" management.

To further aid the server in processing the generated command, additional information such as the time the event was generated, the source node, etc., are also contained in the client's message.

<u>Brief Description of the Drawing</u>

Fig. 1 is a diagram illustrating the overall structure of an MPEG-4 client or terminal.

Fig. 2 shows the MPEG-4 System Decoder Model.

Fig. 3 illustrates the method used in MPEG-4 for associating audio visual objects with their encoded data in other streams via object descriptors and elementary stream descriptors.

Fig. 4 shows a generic configuration of a client/server MPEG-4-based communication system.

Fig. 5 illustrates the method of associating scene description nodes, especially sensor nodes, to command descriptors using: (a) server routes, and (b) command route nodes.

Fig. 6 shows the binary syntax of the command descriptor as described in a preferred embodiment.

Fig. 7 shows the binary syntax of the command descriptor remove command as described in the preferred embodiment.

Fig. 8 shows the binary syntax of the server route structure, and how it is added to the main MPEG-4 BIFS scene syntax.

Fig. 9 shows the node syntax and semantics of the Command Route structure.

Fig. 10 shows an indicative list of predefined Command IDs and their associated interpretation.

Fig. 11 shows the binary syntax of the command contained in a command descriptor for the preferred embodiment.

Fig. 12 is a flow diagram illustrating the process of triggering a server command in an MPEG-4 client, when Server Routes are used.

Fig. 13 is a flow diagram illustrating the process of assembling the data to be placed in the command sent back to the server, when Server Routes are used.

Fig. 14 is a flow diagram illustrating the process of triggering a server command in an MPEG-4 client, when Command Route nodes are used.

Fig. 15 is a flow diagram illustrating the process of assembling the data to be placed in the command sent back to the server, when Command Route nodes are used.

Detailed Description

Reference will now be made in detail to the preferred embodiment of the invention as illustrated in the figures.

MPEG-4 is an international standard being developed under the auspices of the International Standardization Organization (ISO). Its official designation is ISO/IEC 14496. Its basic difference with previous ISO or ITU standards such as MPEG-1, MPEG-2, H. 261, or H. 263 is that is addresses there presentation of audiovisual objects. Thus, the different elements comprising an audiovisual scene are first encoded separately, using techniques that are being defined in Parts 2 (Visual) and 3 (Audio) of the specification. These objects are transmitted to the receiver or read from a mass storage device together with scene description information that describes how these objects are to be placed in space and time in order to be presented to the user.

The coded data for each audiovisual object as well as the scene description information proper are transmitted in their own "channels" or elementary streams. Additional control information is also transmitted, as further discussed below, in order to allow the receiver to correctly associate audio visual objects referenced in the scene with the elementary streams that contain their encoded data.

In order to fully describe the structure of MPEG-4, we refer to Fig. 1. At the bottom of the figure, the various possible delivery systems are shown, including (but not limited to) ATM, IP, MPEG-2 Transport Stream (TS), DVB, either over a communication link or a mass storage device. In contrast to MPEG-2, MPEG-4 does not define its own transport layer facility, in order to allow delivery over a wide variety of communication environments. For delivery systems that may lack appropriate multiplexing capability, e.g. GSM wireless data channels, or that require low delay, a simple multiplexing tool called FlexMux is defined.

This infrastructure is used to deliver to the client a set of elementary streams. The streams contain any one of scene description information, audio visual object data (e.g. coded video, such as an MPEG-2 or MPEG-4 video stream), or control information (namely, object descriptors). Each elementary stream can contain data of

only one type.

The data contained in an elementary stream are packaged according to the MPEG-4 Sync Layer (SL), which packages access units of the underlying medium (e.g., a frame of video or audio, or a scene description command) and adds timing information (clock references and time stamps), sequence numbers, etc. The SL is shown at the middle portion of Fig. 1.

Encoding of individual audiovisual object data is performed according to Parts 2 and 3 of the MPEG-4 specification. It is furthermore allowed to utilize other encodings, such as MPEG-1 or MPEG-2. The scene description as well as the control information (object descriptors) is encoded as defined in Part 1 of MPEG-4.

The receiver processes the scene description information as well as the decoded audiovisual object data and performs composition, i.e. the process of combining the objects together in a single unit, and rendering, i.e. the process of displaying the result in the user's monitor or playing it back in the user's speaker is in the case of audio. This is shown at the top of Fig. 1.

Depending on the information contained in the scene description, the user may have the opportunity to interact with the scene. In addition, the scene description may contain information that enables dynamic behavior. In other words, the scene itself may generate events, without user intervention.

The object-based structure of MPEG-4 necessitated the definition of a more general system decoder model compared with MPEG-2 or other systems. In particular, as shown in Fig. 2, the receiver is considered to be equipped with a set of decoders, one for each object. Each decoder has a decoding buffer, as well as a composition buffer. Decoding buffers are managed by the sender using techniques similar to MPEG-2, i.e., clock references for clock recovery, and decoding time stamps for removal of data from the decoding buffer followed by theoretically instantaneous decoding. Data placed in composition buffers are available for use by the compositor, and overwrite any previously placed data. The decoding buffers are filled by the demultiplexer, which is encapsulated within the DMIF (Digital Media Integration Framework, Part 6 of MPEG-4) Application Interface. This is a conceptual interface, requiring no further description here.

MPEG-4 SCENE DESCRIPTION

The scene description information in MPEG-4 is an extension of VRML 2.0 (Virtual Reality Modeling Language) specification. VRML uses a tree structured approach to define scenes. Each node in the scene performs a composition and/or grouping operation, with the leaves containing the actual visual or audio information. Furthermore, nodes contain fields that affect their behavior. For example, a Transform node contains a Rotation field to define the angle of rotation.

MPEG-4 defines some additional nodes that address 2-D composition, as

VRML is a purely 3-D scene description language. This addresses the needs of applications that focus on low-cost systems. In contrast to VRML, MPEG-4 does not use a text-based scene description but instead defines a bandwidth-efficient compressed representation called BIFS (Binary Format for Scenes).

The BIFS encoding follows closely the textual specification of VRML scenes. In particular, node coding is performed in a depth-first fashion, similarly to text-based VRML files. As in VRML, the fields of each type of node assume default values. Hence only fields that have non-default values need to be specified within each node. Field coding within a node is performed using a simple index-based method followed by the value of the coded field. Node type coding is more complicated. In order to further increase band width efficiency the context of the parent field (if any) is taken into account. Each field that accepts children nodes is associated with a particular node data type. Nodes are then encoded using an index which is particular to this node data type, in known fashion.

Each coded node can also be assigned a node identifier (an integer, typically). This allows the reuse of that node in other places in the scene. This is identical to the USE/DEF mechanism of VRML. More important, however, is the fact that it allows it to participate in the interaction process.

The interaction model used in MPEG-4 is the same as in VRML. In particular, fields of a node can act as event sources, event sinks, or both. An event source is associated with a particular user action or system event. Example of user event are sensor nodes that can detect when the mouse has been clicked. Example of system events are timers (TimeSensor node) that are triggered according to the system's time.

Dynamic scene behavior and interactivity are effected by linking event source fields to event sink fields. The actual linking is performed using the mechanism of ROUTEs. MPEG-4 route specifications, if any, are given immediately after the scene node descriptions. Their encoding is based on the node identifiers for the source and sink nodes, as well as the field indices of the source and sink fields.

An important distinction between VRML and MPEG-4 is that in the latter, scene descriptions can be updated dynamically using time-stamped commands. In Contrast, VRML operates on static "worlds". After a world is loaded, there is no mechanism to modify it. In MPEG-4, objects can be added or deleted, and parts of the scene (or the entire scene) can be replaced.

OBJECT DESCRIPTORS

In order to have a very flexible structure (facilitating editing, etc.), the actual content.of audiovisual objects is not contained within the scene description itself. In other words, BIFS only provides the information for the scene structure, as well as objects that are purely synthetic, e.g. a red rectangle that is constructed using

BIFS/VRML nodes. Audio visual objects that require coded information are represented in the scene by leaf nodes which either point to a URL or an object descriptor.

Object descriptors are hierarchically structured information that describe the elementary streams that comprise the coded representation of a single audio visual object. More than one stream may be required, e.g. for stereo or multi-language audio, or hierarchically coded video. The topmost descriptor is called object descriptor, and is a shell that is used to associate an object descriptor identifier (OD-ID) with a set of elementary stream descriptors. The latter contain an ES-ID as well as information about the type of data contained in the elementary stream associated with the particular ES-ID. This information tells the receiver, e.g, that a stream contains MPEG-2 Video data, following the Mail Profile at the Main Level.

The mapping of the ES-ID to an actual elementary stream is performed using a stream map table. For example, it may associate ES-ID 10 with support number 1025. This table is made available to the receiver during session set up. The use of multiple levels of indirection facilitates manipulation of MPEG-4 content. For example, remultiplexing would only require a different stream map table. No other information would have to be modified within the MPEG-4 content.

Object descriptors are transmitted in their own elementary streams, and are packaged in commands according to the Sync Layer syntax. This allows object descriptors to be updated, added, or removed.

Fig. 3 depicts the process with which object descriptors and elementary stream descriptors are used to associate audiovisual objects in the scene description with their elementary streams. First, a special Initial Object Descriptor is used to bootstrap the MPEG-4 receiver by pointing to the object descriptor stream and the scene descriptor stream associated with the selected content. This descriptor is delivered to the receiver during session set up.

The scene description in this example contains an Audio Source node, which points to one of the object descriptors. The descriptor, in turn, contains an elementary stream descriptor that provides the ES-ID for the associated stream. The ES-ID is resolved to an actual transport channel using the stream map table. The scene also has a Movie Texture node that, in this case, uses scalable video with two streams. As a result, two elementary stream descriptors are contained, each pointing to the appropriate stream (base and enhancement layer).

MPEG-4 CLIENT/SERVER INTERACTION

From the preceding description, and considering the MPEG-4/VRML scene description framework, it is evident that while a rich local interaction framework is provided, there is no facility to effect server-based interaction. In particular, there is no mechanism with which to either describe messages that are to be sent back to a

server, or trigger the generation of such messages.

Fig. 4 depicts an example client/server environment. On the left side of the figure there is an MPEG-4 server, including a pump that performs timed transmission of data read as SL-packetized streams, and an instance of a DMIF service provider, in this case utilizing the MPEG-4FlexMux multiplexing tool.

The use of FlexMux is optional. Other server structures can be used, as known in the art. For example, the data source could be an MPEG-4 file instead of SL-packetized streams.

The information generated at the server is sent across a network (e. g. IP or ATM) to the receiver. On the receiving side, we have a similar instance of a DMIF service provider which delivers demultiplexed elementary streams to the player. The DMIF-to-DMIF signaling is one method of performing session set up, and is described in Part 6 of MPEG-4. Other methods are possible, as known in the art, including Internet-based protocols such as SIP, SDP, RTSP, etc.

One of the main objectives of the present invention is a process with which server commands can be first described and transmitted from the server to the client, then triggered at the client at the appropriate times, and finally sent back to the server in order to initiate further action.

COMMAND DESCRIPTORS

The Command Descriptor framework provides a means for associating commands with event sources within the nodes of a scene graph. When a user interacts with the scene, the associated event is triggered and the commands are subsequently processed and transmitted back to the server.

The actual interaction itself is specified by the content creator and may be a mouse click or mouse-over or some other form of interaction (e.g., a system event).

The command descriptor framework consists of three elements, a Command Descriptor, a server route or Command Route node, and a Command.

A Command Descriptor contains an ID (an integer identifier) as well as the actual command that will eventually be transmitted back to the server if and when an associated event is triggered.

The ID is used to refer to this command descriptor from the scene description information. By separating the command descriptor and the command it contains from the scene itself, we allow for more than one event to use the same command. This also permits modification of the command without changing the scene description in any way.

The association of the Command Descriptors to the event source field node can be performed in different ways.

First, a Server Route can be added to the regular route facility of BIFS. The difference with traditional routes is that the target of the route is not another field, but

rather the command descriptor. This structure is depicted in Fig. 5(a). In this example we have two nodes in a scene tree fielding events to two command descriptors. A sensor node triggers an event to another node with an event source, event out, or event source/sink, exposed Field in MPEG-4 terminology, which in turn is routed via a server route to Command Descriptor 1. For Command Descriptor 2, there are direct server routes from the nodes to the descriptor.

A second approach consists of adding a new Command Route node type to the list of nodes supported by MPEG-4. This node has an 'execute' event sink field, as well as a field containing the command descriptor ID. Whenever the 'execute' field receives an event, using regular MPEG-4/VRML routes, the command descriptor associated with that ID is used to issue a command back to the server. This structure is depicted in Fig. 5(b). As compared with Fig. 5(a), the Server Routes are substituted with Command Route nodes. The operation in the two cases is essentially the same.

The Command Descriptor syntax is shown in Fig. 6. We use the Flavor media representation language to describe the bit stream syntax, which is also used in Part 1 of the MPEG-4 specification (see www.ee.columbia.edu/flavor or Part 1 of MPEG-4). The command descriptor begins with a special tag that identifies it as a descriptor of this particular type. We then have the descriptor ID, followed by a command ID. The latter is used to signal predefined server commands, such as "start", "pause", or "stop". Following that, we have the length indication of the remaining data in the descriptor, counted in bytes. Then, a count of the number of ES-IDs that will be provided to transmit the message back to the server(s). More than one is given in case we want to effect one-to-many communication, i.e. a single command to be communicated to multiple servers. This is followed by the series of the desired ES-IDs. Finally, a set of application-specific parameters are included. These will be passed back to the server when the command is triggered. Depending on the value of the command ID, the semantics of these parameters may be predefined.

The byte-oriented structure of the Command Descriptor allows it to be very easily generated on-the-fly by the server. This is an important feature for applications that involve "cookie" management, among others, where the command parameters need to be continuously updated by the server after processing each user event.

In order to update a Command Descriptor, the server or content creator only needs to submit a new one with the same ID.

In order to remove a Command Descriptor, a special command is provided. The syntax is shown in Fig. 7. The command begins with a tag that identifies this descriptor as a Command Descriptor Remove command. It is then followed by the ID of the Command Descriptor to be removed.

Both the Command Descriptor and Command Descriptor Remove structures can be carried within the object descriptor stream. Due to the structure of the object descriptor framework, using tags to identify descriptors, these can be interspersed

with other descriptors.

As mentioned above, there are two ways to link the Command Descriptors to the scene. The first one relies on server routes. These require an extension of the bit stream syntax of the scene description of MPEG-4 as shown in Fig. 8. In particular, the main BIFS Scene structure is extended to include Server ROUTE structures. The only difference between a regular ROUTE and a Server ROUTE is that instead of a target node/field the ID of the target command descriptor is specified. It is known to those skilled in the art how to modify other ROUTE commands (insertion, deletion, etc.) in order to accommodate Server ROUTEs. In all cases, the target node/field pair needs to be changed to indicate the target Command Descriptor.

Using the Command Route node approach, a new node type needs to be defined. The node definition is provided in Fig. 9, using the standard node definition table used in Part 1 of the MPEG-4 specification. The node contains only two fields, namely an 'execute' field that serves as an event sink, and a 'command Descriptor' field which contains either a URL pointing to a Command Descriptor or the ID of the Command Descriptor to be associated with this Command Route node. As with all SFUrl fields, selection between ID and URL is performed using a one-bit flag (SFUrl field encoding is defined in Part 1 of MPEG-4) .

Using the Command Route node approach, changes to the association of events with Command Descriptors can be performed using the standard MPEG-4 BIFS commands for changing nodes and fields within them.

Command descriptors can be updated using command descriptor updates allowing a node to support different interaction behavior at different times depending on the command in the command descriptor. The commands are transmitted to the server using the DAI user command primitives supported by DMIF.

It is also possible to extend the command descriptor to include the protocols used to transmit the commands. For example a different tag can be used to indicate submission using HTTP POST/GET, rather than the standard MPEG-4 facilities.

Even though the command descriptor framework is generic and supports application-specific user interaction, a few standard commands are needed in order to support consistent application behavior across applications and servers. This is particularly true for common commands such as stream control. As a result, we specify a set of common control commands. A server should be aware of any application specific commands in order to process them. The set Of standard commands allows interoperation with any server. Fig. 10 shows a set of commands together with their command IDs for this embodiment. Other assignments are also possible, as is evident to persons skilled in the art.

Fig. 11 depicts the syntax of the command as it is sent from the client to the server. It is essentially a copy of the Command Descriptor, minus the descriptor ID. In particular, it contains the Command ID, the set of ES-IDs to which this command

was transmitted, as well as the set of parameters that were specified in the Command Descriptor. These commands are transmitted in SL-packetized streams, and hence full timing and sequencing information can be made available to the server.

PROCESSING EVENTS FOR DISPATCHING SERVER COMMANDS

We now discuss in detail the process of generating commands based on user or system events, starting with the use of Server ROUTEs.

Referring to Fig. 12, upon the generation of a user or system event, the receiver propagates the event through the network of ROUTEs and Server ROUTEs. For the purposes of event propagation, the type of ROUTE does not matter, so that the same algorithm can be used. If an event is propagated through a Server Route, the system checks if that event corresponds to a condition associated with the logical True value. If no, the server command processing terminates; yes, then the dispatch process is executed.

This process, for the Server Route case, is depicted in Fig. 13. The process obtains the Command Descriptor ID from the Server Route. It then correlates it with the information it has on the Command Descriptors available in tie scene. If no match is found, this is an error and no further action is taken. If a match is found, then the system examines the command ID in order to see if it corresponds to known semantics (pre-defined command IDs). If it corresponds to known semantics then the system may process the command parameters according to the desired semantics. If it does not correspond to known semantics then the system skips this state, and directly packages the indicated Command and transmits it to the server.

In the case of Command Route nodes, referring to Fig. 14, upon the generation of a user or system event, the receiver propagates the event through the network of ROUTEs. If an event reached the 'execute' field of a Command Route node, the system check if that event corresponds to a condition associated with the logical True value. If no, server command processing terminates; if yes, the dispatch process is executed.

This process, for the Command Route node case, is depicted in Fig. 15. The sequence of steps is essentially identical to that of Fig. 13, the difference being that the reference for the Command Descriptor ID is now in the Command Route node rather than a Server Route.